CS 271 - Project 0011

James Le

February 15, 2017

1. Where in a min-heap might the largest element reside, assuming that all elements are distinct?

   Since the parent is greater or equal to its children, the smallest element must be a leaf node.

2. Is an array in sorted order a min-heap? Why or why not?

   An array sorted from lowest to highest is a min-heap when using the array-based heap implementation. The heap property that the parent node is greater than its child nodes (2i + 1 and 2i + 2, using zero-based arrays) holds for all nodes that have children.

3. Implement a MinHeap template class, and the heap sort algorithm. The heap sort method should return a copy of the array in ascending sorted order.

4. Plot the running time of heap sort for a variety of sufficiently large input sizes and compare this to the running time of the other sorting algorithms that you implemented in CS 173.

   Heap Sort

   - Best Case: $\Omega(n \log(n))$
   - Average Case: $\Theta(n \log(n))$
   - Worst Case: $O(n \log(n))$

   Quick Sort

   - Best Case: $\Omega(n \log(n))$
   - Average Case: $\Theta(n \log(n))$
   - Worst Case: $O(n^2)$

   Merge Sort

   - Best Case: $\Omega(n \log(n))$
   - Average Case: $\Theta(n \log(n))$
   - Worst Case: $O(n \log(n))$

   Bubble Sort

   - Best Case: $\Omega(n)$
   - Average Case: $\Theta(n^2)$
   - Worst Case: $O(n^2)$

   Insertion Sort

   - Best Case: $\Omega(n)$
   - Average Case: $\Theta(n^2)$
   - Worst Case: $O(n^2)$

   Selection Sort

   - Best Case: $\Omega(n^2)$
   - Average Case: $\Theta(n^2)$
   - Worst Case: $O(n^2)$

5. What is the asymptotic time complexity of the heap sort algorithm on an array that is already sorted? What is the asymptotic time complexity on an array that is in reverse order? What is the best case asymptotic time complexity of heap sort, and on what kind of input does it occur?

   - On an array that is already sorted, the asymptotic time complexity of heap sort algorithm is $O(1)$.
   - On an array that is in reverse order (worst case), the asymptotic time complexity of heap sort algorithm is $O(n \log n)$.
   - The best case asymptotic time complexity of heap sort is $\Omega(n \log n)$.

```
// James Le - CS 271
// Feb 15, 2017
// heap.h
// a binary min heap

#ifndef HEAP_H
#define HEAP_H

#include <iostream>

const int DEFAULT_SIZE = 100;

template <class KeyType>
class MinHeap
{
  public:
    MinHeap(int n = DEFAULT_SIZE);          // default constructor
    MinHeap(KeyType initA[], int n);        // construct heap from array
    MinHeap(const MinHeap<KeyType>& heap);  // copy constructor
    ~MinHeap();                             // destructor

    void heapSort(KeyType sorted[]);  // heapsort, return result in sorted

    MinHeap<KeyType>& operator=(const MinHeap<KeyType>& heap);  // assignment operator
    std::string toString() const;     // return string representation

  protected:
    KeyType *A;     // array containing the heap
    int heapSize;   // size of the heap
    int capacity;   // size of A

          void heapify(int index);          // heapify subheap rooted at index
    void buildHeap();                 // build heap
        int leftChild(int index) { return 2 * index + 1; }  // return index of left child
        int rightChild(int index) { return 2 * index + 2; } // return index of right chil
d
        int parent(int index) { return (index - 1) / 2; }   // return index of parent
    void swap(int index1, int index2);        // swap elements in A
    void copy(const MinHeap<KeyType>& heap);  // copy heap to this heap
    void destroy();                           // deallocate heap
};

template <class KeyType>
std::ostream& operator<<(std::ostream& stream, const MinHeap<KeyType>& heap);

#include "heap.cpp"

#endif
```

```
// James Le - CS 271
// Feb 15, 2017
// heap.cpp
// C++ program for implementation of a MinHeap template class and Heap Sort algorithm

#include <iostream>
#include <cmath>
#include <ctime>
#include <string>
#include <sstream>
#include <cassert>

using namespace std;

/*================================================================
MinHeap Default Constructor
Pre: None
Post: Construct a new MinHeap data type with default size set to n
================================================================*/
template <class KeyType>
MinHeap<KeyType>::MinHeap(int n)
{
  heapSize = 0;
  capacity = n;
  A = new KeyType[capacity];
}
/*================================================================================
MinHeap Array Constructor
Pre: An empty array initA with size n
Post: Construct a heap from array initA with contents from the heap from array A
================================================================================*/
template <class KeyType>
MinHeap<KeyType>::MinHeap(KeyType initA[], int n)
{
  capacity = n;
  A = new KeyType[capacity];
  heapSize = 0;

  // Copy the array into the heap's internal array
  for (int i = 0; i < n; i++)
    A[i] = initA[i];

  heapSize = capacity;
  // Organize the array into a proper MinHeap Tree
  buildHeap();
}

/*============================================================
MinHeap Copy Constructor
Pre: heap is a MinHeap data type
Post: Construct a MinHeap data type with same contents as heap
============================================================*/
template <class KeyType>
MinHeap<KeyType>::MinHeap(const MinHeap<KeyType>& heap)
{
  copy(heap);
}

/*=====================================================
MinHeap Destructor
Pre: None
Post: Clean up the MinHeap class
=====================================================*/
template <class KeyType>
MinHeap<KeyType>::~MinHeap()
{
  destroy();
}

/*=================================================
```

```
Assignment Operator
Pre: heap is a MinHeap data type
Post: Allow assignment of values between MinHeap class
==================================================*/
template<class KeyType>
MinHeap<KeyType>& MinHeap<KeyType>::operator=(const MinHeap<KeyType>& heap)
{
  if(this != &heap)
  {
    delete [] A;
    copy(heap);
  }
  return *this;
}

/*==============================
String Representation for MinHeap
==============================*/
template<class KeyType>
std::string MinHeap<KeyType>::toString() const
{
  string heapstr = "[";
  KeyType updatestring;
  for(int i = 0; i < capacity; i++)
  {
    ostringstream change;
    if(i != heapSize - 1)
    {
      if(i != capacity - 1)
      {
        updatestring = A[i];
        change << updatestring;
        heapstr += change.str();
        heapstr += ", ";
      }
      else
      {
        updatestring = A[i];
        change << updatestring;
        heapstr += change.str();
      }
    }
    else if(i == heapSize - 1)
    {
      updatestring = A[i];
      change << updatestring;
      heapstr += change.str() + " |";
    }
  }
  heapstr += "]";
  return heapstr;
}

/*==========================
Heapsort Algorithm
Pre: a list of Key Types
Post: sorted list of Key Types
==========================*/
template <class KeyType>
void MinHeap<KeyType>::heapSort(KeyType sorted[])
{
  // One by one extract an element from heap
  for (int i = heapSize - 1; i >= 0; i--)
  {
    // Move current root to the end
    swap(0,i);
    heapSize--;
    // Call min-heapify on the reduced heap
    heapify(0);
  }
```

```
  for (int j = 0; j < capacity; j++)
  {
    sorted[(capacity - 1) - j] = A[j];
  }

  // Copy elements in A to sorted
  for (int l = 0; l < capacity; l++)
  {
    A[l] = sorted[l];
  }
}

/*=============================================
Min-Heapify Subheap Rooted At Index
Pre: 2 subtrees for the children are already heaps
Post: Root A[0] is the root of the heap
=============================================*/
template <class KeyType>
void MinHeap<KeyType>::heapify(int index)
{
  int smallest = index; // Initialize smallest as root
  int l = leftChild(index);
  int r = rightChild(index);

  // If left child is smaller than root
  if (l < heapSize && A[l] < A[smallest])
    smallest = l;

  // If right child is smaller than smallest so far
  if (r < heapSize && A[r] < A[smallest])
    smallest = r;

  // If smallest is not root
  if (smallest != index)
  {
    swap(index, smallest);
    // Recursively heapify the affected sub-tree
    heapify(smallest);
  }
}

/*=========================================
Build MinHeap
Pre: None
Post: A heap that is sorted in decreasing order
=========================================*/
template <class KeyType>
void MinHeap<KeyType>::buildHeap()
{
  for (int i = ((capacity/2) - 1); i >= 0; i--)
  {
    heapify(i);
  }
}

/*=============================================
Swap Elements
Pre: 2 elements
Post: The 2 elements' index positions are swapped
=============================================*/
template <class KeyType>
void MinHeap<KeyType>::swap(int index1, int index2)
{
  KeyType temp = A[index1];
  A[index1] = A[index2];
  A[index2] = temp;
}

/*=============================================================
```

```
Copy Heap to another Heap
Pre: heap is a MinHeap data type
Post: Construct a MinHeap data type with same contents as heap
============================================================*/
template <class KeyType>
void MinHeap<KeyType>::copy(const MinHeap<KeyType>& heap)
{
  heapSize = heap.heapSize;
  capacity = heap.capacity;

  A = new KeyType[heap.capacity];
  for (int i = 0; i < heap.heapSize; i++)
  {
    A[i] = heap.A[i];
  }
}

/*============================================
Deallocate Heap
Pre: None
Post: Clean up the MinHeap class
============================================*/
template <class KeyType>
void MinHeap<KeyType>::destroy()
{
  delete [] A;
  heapSize = 0;
  capacity = 0;
}

/*====================================================================
Overloading Operator
Pre: When all the heap or array is turned into a string
Post: When all of the heap is converted to a string
====================================================================*/
template <class KeyType>
std::ostream& operator<<(std::ostream& stream, const MinHeap<KeyType>& heap)
{
  stream << heap.toString();
  return stream;
}
```

```
// James Le - CS 271
// Feb 15, 2017
// test_heap.cpp
// a binary min heap

#include <sys/time.h>
#include "heap.h"

using namespace std;

void TestToString()
{
  float A[5] = {1.3,5.5,3.7,8.9,6.2};
  MinHeap<float> astro1(A,5);
  assert(astro1.toString() == "[1.3, 5.5, 3.7, 8.9, 6.2 |]");

  long B[5] = {234243,45345345,91872214465,1983275,9876};
  MinHeap<long> astro2(B,5);
  assert(astro2.toString() == "[9876, 234243, 91872214465, 1983275, 45345345 |]");

  int C[11] = {5,2,7,9,56,76,34,64,23,4,8};
  MinHeap<int> astro3(C,11);
  assert(astro3.toString() == "[2, 4, 7, 9, 5, 76, 34, 64, 23, 56, 8 |]");

  double D[21] = {3,445,667,9,56,76,34,64,24,8,5,2,7,9,56,76,34,64,23,4,9};
  MinHeap<double> astro4(D,21);
  assert(astro4.toString() == "[2, 4, 3, 9, 5, 7, 9, 34, 23, 8, 445, 76, 667, 34, 56, 76,
 64, 64, 24, 56, 9 |]");
}

void TestHeapSort()
{
  float A[5] = {1.3,5.5,3.7,8.9,6.2};
  MinHeap<float> astro1(A,5);
  astro1.heapSort(A);
  assert(astro1.toString() == "[1.3, 3.7, 5.5, 6.2, 8.9]");

  long B[5] = {234243,45345345,91872214465,1983275,9876};
  MinHeap<long> astro2(B,5);
  astro2.heapSort(B);
  assert(astro2.toString() == "[9876, 234243, 1983275, 45345345, 91872214465]");

  int C[11] = {5,2,7,9,56,76,34,64,23,4,8};
  MinHeap<int> astro3(C,11);
  astro3.heapSort(C);
  assert(astro3.toString() == "[2, 4, 5, 7, 8, 9, 23, 34, 56, 64, 76]");

  double D[21] = {3,445,667,9,56,76,34,64,24,8,5,2,7,9,56,76,34,64,23,4,9};
  MinHeap<double> astro4(D,21);
  astro4.heapSort(D);
  assert(astro4.toString() == "[2, 3, 4, 5, 7, 8, 9, 9, 9, 23, 24, 34, 34, 56, 56, 64, 64
, 76, 76, 445, 667]");
}

void TestOperator()
{
  float A[11] = {5.1,2.1,7.1,9.1,56.1,76.1,34.1,64.1,23.1,4.1,8.1};
  MinHeap<float> astro1(A,11);
  float B[11] = {3.1,445.1,667.1,9.1,56.1,76.1,34.1,64.1,23.1,4.1,8.1};
  MinHeap<float> astro2(B,11);
  assert(astro1.toString() == "[2.1, 4.1, 7.1, 9.1, 5.1, 76.1, 34.1, 64.1, 23.1, 56.1, 8.
1 |]");
  assert(astro2.toString() == "[3.1, 4.1, 34.1, 9.1, 8.1, 76.1, 667.1, 64.1, 23.1, 56.1,
445.1 |]");
  astro1 = astro2;
  assert(astro1.toString() == "[3.1, 4.1, 34.1, 9.1, 8.1, 76.1, 667.1, 64.1, 23.1, 56.1,
445.1 |]");

  long C[5] = {234243,45345345,91872214465,1983275,9876};
  MinHeap<long> astro3(C,5);
```

```
  long D[5] = {1234243,145345345,191872214465,11983275,19876};
  MinHeap<long> astro4(D,5);
  assert(astro3.toString() == "[9876, 234243, 91872214465, 1983275, 45345345 |]");
  assert(astro4.toString() == "[19876, 1234243, 191872214465, 11983275, 145345345 |]");
  astro3 = astro4;
  assert(astro3.toString() == "[19876, 1234243, 191872214465, 11983275, 145345345 |]");
  assert(astro4.toString() == "[19876, 1234243, 191872214465, 11983275, 145345345 |]");

  int E[11] = {5,2,7,9,56,76,34,64,23,4,8};
  MinHeap<int> astro5(E,11);
  int F[11] = {3,445,667,9,56,76,34,64,23,4,8};
  MinHeap<int> astro6(F,11);
  assert(astro5.toString() == "[2, 4, 7, 9, 5, 76, 34, 64, 23, 56, 8 |]");
  assert(astro6.toString() == "[3, 4, 34, 9, 8, 76, 667, 64, 23, 56, 445 |]");
  astro5 = astro6;
  assert(astro5.toString() == "[3, 4, 34, 9, 8, 76, 667, 64, 23, 56, 445 |]");
  assert(astro6.toString() == "[3, 4, 34, 9, 8, 76, 667, 64, 23, 56, 445 |]");

  double G[21] = {3,445,667,9,56,76,34,64,24,8,5,2,7,9,56,76,34,64,23,4,9};
  MinHeap<double> astro7(G,21);
  double H[21] = {31,4145,6167,199,516,716,314,614,214,81,51,21,71,91,5116,716,314,614,21
3,41,19};
  MinHeap<double> astro8(H,21);
  assert(astro7.toString() == "[2, 4, 3, 9, 5, 7, 9, 34, 23, 8, 445, 76, 667, 34, 56, 76,
 64, 64, 24, 56, 9 |]");
  assert(astro8.toString() == "[19, 31, 21, 199, 41, 71, 91, 314, 213, 81, 51, 716, 6167,
 314, 5116, 716, 614, 614, 214, 516, 4145 |]");
  astro7 = astro8;
  assert(astro7.toString() == "[19, 31, 21, 199, 41, 71, 91, 314, 213, 81, 51, 716, 6167,
 314, 5116, 716, 614, 614, 214, 516, 4145 |]");
  assert(astro8.toString() == "[19, 31, 21, 199, 41, 71, 91, 314, 213, 81, 51, 716, 6167,
 314, 5116, 716, 614, 614, 214, 516, 4145 |]");
}

void TestTime()
{
  timeval timeBefore, timeAfter; // timeval type defined in sys/time.h
  long diffSeconds, diffUSeconds; // elapsed seconds and microseconds
  int A[10] = {10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000};

  for (int i = 0; i < 10; i++)
  {
    int q = A[i];
    int a[q];
    int b[q];
    srand(time(NULL));
    for(int i = 0; i < q; i++)
    {
      a[i] = rand() % 100 + 1;
    }
    MinHeap<int> TheSheriff(a, q);
    int B[q];
    gettimeofday(&timeBefore, NULL); // get the time before
    TheSheriff.heapSort(B);
    gettimeofday(&timeAfter, NULL); // get the time after
    diffSeconds = timeAfter.tv_sec - timeBefore.tv_sec; // elapsed seconds
    diffUSeconds = timeAfter.tv_usec - timeBefore.tv_usec; // elapsed microseconds
    double time = diffSeconds + diffUSeconds / 100000.0; // total elapsed time
  }
}

int main()
{
  TestTime();
  TestToString();
  TestHeapSort();
  TestOperator();

  return 0;
}
```